An Analysis of the Manufacturing Messaging Specification Protocol

Jan Tore Sørensen[1] and Martin Gilje Jaatun[2]

[1] mnemonic as, NO-0167 Oslo, Norway
[2] SINTEF ICT, NO-7465 Trondheim, Norway
jantore@mnemonic.no

Abstract. The Manufacturing Messaging Specification (MMS) protocol is widely used in industrial process control applications, but it is poorly documented. In this paper we present an analysis of the MMS protocol in order to improve understanding of MMS in the context of information security. Our findings show that MMS has insufficient security mechanisms, and the meagre security mechanisms that are available are not implemented in commercially available industrial devices.

Keywords: Process control; Industrial Networks; Protocols; Security.

1 Introduction

We will in this paper present an analysis of the Manufacturing Messaging Specification (MMS[1]) protocol and propose improvements based on our findings. MMS is defined in the ISO 9506 standard [1], and there is also some information available in white-papers from SISCO[2] [2,3,4]. We have tested the MMS protocol as implemented in a major brand of controller used in the process control industry. We will in the following look closer into the construction of MMS packages and how they may be altered and forged.

MMS is an application-layer protocol which specifies services for exchange of real-time data and supervisory control information between networked devices and/or computer applications. It is designed to provide a generic messaging system for communication between heterogeneous industrial devices, and the specification only describes the network-visible aspects of communication. By choosing this strategy, the MMS does not specify the internal workings of an entity, only the communication between a client and a server, allowing vendors full flexibility in their implementation. In order to provide this independence, the MMS defines a complete communication mechanism between entities, composed of [3]:

1. **Objects:** A set of standard objects which must exist in every conformant device, on which operations can be executed (examples: read and write local variables, signal events).

[1] In this paper, MMS does *not* stand for Multimedia Messaging Service, as is often the case elsewhere.

[2] System Integration Specialists Company - not to be confused with Cisco.

F.E. Sandnes et al. (Eds.): UIC 2008, LNCS 5061, pp. 602–615, 2008.

2. **Messages:** A set of standard messages exchanged between a client and a server station for the purpose of controlling these objects
3. **Encoding Rules:** A set of encoding rules for these messages (how values and parameters are mapped to bits and bytes when transmitted)
4. **Protocol**: A set of protocols (rules for exchanging messages between devices).

MMS composes a model from the definition of *objects*, *services* and *behavior* named the Virtual Manufacturing Device (VMD) Model. The VMD uses an object-oriented approach to represent different physical industrial (real) devices in a generic manner. Some of these objects are variables, variable type definitions, programs, events, historical logs (called journals) and semaphores. Along with the definition of these objects, MMS defines a set of communications services that an application can use to manipulate these objects.

We observe that in the literature the terms *services*, *service primitives* and *messages* are all used to describe the functions that manipulate objects or their attributes. We will therefore in this paper use the term service primitive as this is used in the ISO 9506 standard [1], unless we are citing directly from a written source, in which case the quote will be evident in the text. The standard also refers to physical industrial devices as "real devices" and we will continue to use this terminology to avoid confusion.

As MMS is based on an object oriented approach, we will give a brief introduction to the addressing and object hierarchy of MMS, before focusing on the network communication.

1.1 Architecture and Addressing

The MMS architecture is based on a common client server model. Real devices used in industrial networks often contain an MMS server allowing the device to be monitored and managed from an MMS client. An MMS client is typically part of an control builder application or an MMS-to-OPC gateway (MMS/OPC GW). A control builder is an application used to program and monitor industrial controllers. Both the control builder and the MMS/OPC GW use service primitives provided in the MMS to manage devices containing MMS servers. This is depicted in Fig. 1 [2].

As MMS does not specify how to address clients and servers, an entity containing an MMS client or server must rely on the addressing scheme of underlying protocols in the process of establishing an application association to support the MMS environment[1]. In practice, clients and servers are addressed by their IP address and the MMS server uses port number 102. The addressing allows for an MMS context to be negotiated between two peer applications.

To address an MMS object variable, MMS provides several different address modes. MMS allows an address to have different syntax, based on the implementer's choice of what is most appropriate for that device. The specification separates between *named* and *unnamed* variables.

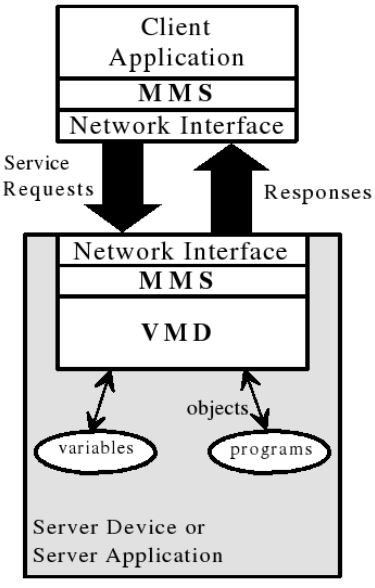

Fig. 1. The VMD model depicting communication between an MMS client and an MMS server

1.2 MMS Objects, Services Primitives and Access Control

Associated with each object is a set of variables that describe values in a given instance of the object. For each object there are corresponding MMS service primitives that allow client applications to access and manipulate those objects. The top level object in the MMS is the VMD which has at least one network-visible address.

Each real device is represented by a real object with vendor-specific features associated with them. The VMD model maps the real object and devices onto virtual objects and devices, described in a generic manner which is in conformance to the VMD model. In other words, a real variable is an element of *typed* data that is contained within a VMD object. An MMS variable is part of a virtual object that represents a mechanism for the MMS client to access the real variable. The MMS server containing the virtual MMS object can be understood as a communication driver which hides the specifics of a real device from the client. From the client's point of view the virtual MMS variable represents a pointer or an access method to the real variable and it is only the MMS server with its objects and its behavior that is visible to the client. The MMS client can never interact with real device variables directly.

All MMS objects contain an access method variable. This attribute contains the information which a device needs to identify the real variable as described above. It contains values which are necessary to find the memory location of the real variable with the contents that lie outside MMS. A special method, the method *PUBLIC*, is standardized for accessing the real variables.

Table 1. The basic methods inherited from the VMD object

MMS General methods	Description
Get	This method is used to obtain the value of a specified object.
Set	This method is used to write/put value or contents into a specified object.
Query Attributes	This method is used to obtain structure or capability information of a specified object.
Create	This method allows objects of particular classes to be instantiated.
Rename	This method allows instantiated objects to be renamed.
Delete	This method allows instantiated objects to be destroyed.

For each object there are corresponding MMS service primitives that allow client applications to access and manipulate those objects. The MMS defines the service primitives of both clients and servers, but the VMD focuses only on specifying the network-visible behavior of MMS servers. And thus, each vendor of an MMS server device is responsible for hiding vendor specific details of the real objects and devices by providing an *executive function* which maps the real entities up to the virtual level, which shall comply with the VMD model definitions. To ensure vendor implementation compliance with the VMD model, it specifies how MMS devices containing an MMS server shall provide a consistent and well-defined view of the object contained in the VMD. And thus, MMS provides a common interface for communication with different devices through the generic virtual objects.

All MMS objects except the Operator Station object inherit six abstract services from the VMD object. These are depicted and described in table 1. E.g. service primitives *read* and *RequestDomainUpload* for the objects *Named Variable List* and *Domain* respectively inherit from the abstract service primitive *get*.

MMS uses access control lists to provide explicit control of the ability to access or alter MMS objects. Protection requirements for an MMS variable are inherited from the underlying real variable in the real device. These requirements are established by the access method in the MMS object. ISO 9506 [1] states that each object within an MMS implementation must contain a reference to an *Access Control List* object that specifies the conditions under which services directed at the named object may succeed. For the purposes of specifying the control conditions, services are grouped into six classes as described in table 1. Access control is enforced through special mechanisms provided by MMS. These mechanisms include possession of a semaphore, identity of user (Application Reference), and the submission of a password (which may be arbitrarily complex).

1.3 Network Services

As we have stated earlier, MMS is not by itself a communication protocol, as it only defines messages that have to be transported by an unspecified network.

MMS was originally developed as a part of the MAP specification [5], and is therefore specified on all seven OSI layers as depicted in figure 3. MAP was originally created by General Motors as an internal standard for communications in industrial automation networks. It is now a public, multivendor communications standard for industrial automation equipment. MMS supports the use of both confirmed and unconfirmed services, but we will in this paper focus on the confirmed services. The MMS defines the following Protocol Data Unit (PDUs) for a confirmed service exchange:

- Confirmed-RequestPDU
- Confirmed-ResponsePDU
- Confirmed-ErrorPDU
- Cancel-RequestPDU
- Cancel-ResponsePDU
- Cancel-ErrorPDU
- RejectPDU

These messages will be used in the communication between a MMSclient and server. When a client wishes to invoke a service primitive on the server side application (the VMD), the transitions depicted in Fig. 2 may occur, depending on the responce from the VMD.

Before a service primitive is called through a *Confirmed-RequestPDU*, the server is in a *Responder Idle* state. Upon receipt of a *Confirmed-RequestPDU* (1) for any of the confirmed services, the MMS-provider issues an indication primitive to the application, specifying the particular service being requested and an *invoke ID* that specifies the service instance and enters the state *Service Pending Responder*. Upon receipt of a response service primitive, from the overlaying application, containing a result parameter specifying the service previously indicated and an *invoke ID* that identifies the service instance, the MMS-provider sends a *Confirmed-ResponsePDU* (2) which specifies the service type and the invoke ID from the response primitive along with the requested data. Then a state transition into the *Responder Idle* state occurs.

If the application can not provide the requested data, the MMS-layer will receive a response service primitive containing a Result parameter specifying an errorstate along with the *invoke ID* used to identify the the requesting MMS service instance. The MMS-layer then sends a *Confirmed-ErrorPDU* (3) containing the service type and the *invoke ID* from the response primitive. When sending the *Confirmed-ErrorPDU* a state transition back into the *Responder Idle* state occurs.

Upon receipt of a *Cancel-RequestPDU* from the client containing the *invoke ID* of the matching service instance request, the MMS-layer on the server issues a cancel indication service primitive to the overlaying application. This indication specifies the *invoke ID* of the service request to be canceled; this information is obtained from the Cancel-RequestPDU parameters. The state *Canceling Service Responder* (4) is then entered. Now two things may occur on the server side: 1) If the application has provided the requested data, the server sends a

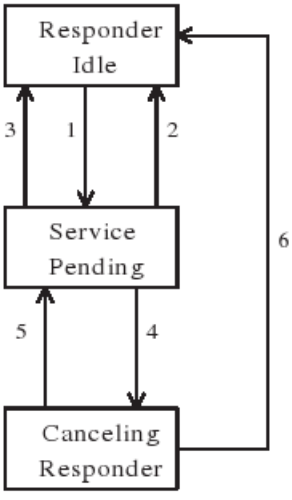

Fig. 2. The MMS Confirmed Service Request as seen by the Service Responder (server)

Cancel-ResponsePDU containing the data and *Confirmed-ErrorPDU* entering the *Responder Idle* state through transition 6 (T6). 2) If the application has not provided the requested data the server sends a *Cancel-ResponsePDU* without any data and enters the *Service Pending Responder* state through T5. The server then sends a *Confirmed-ErrorPDU* through T3 and returns to the *Responder Idle* state.

According to [1] the MMS runs on the network stack depicted in Fig. 3. Like all ISO standards this network stack relates to the Open System Interconnection stack describing the abstract service layers such as session and presentation layer. We will now give a short description of some of the protocols/layers based on their relevance to this paper theme.

2 ASN.1

MMS uses ASN.1 to encode data at the OSI presentation layer before transmission. The ASN.1 representation of data is independent of machine-oriented structures and encodings and also of the physical representation of the data (referred to as transfer syntax in communication terminology). MMS uses BER to encode ASN.1 data before transmission. As we will be decoding BER code, we will explain BER encoding in the next section.

3 BER

The Basic Encoding Rules are one of the original encoding rules specified by the ASN.1 standard for encoding abstract information into a concrete data stream. The rules, collectively referred to as a transfer syntax in ASN.1 parlance, specify

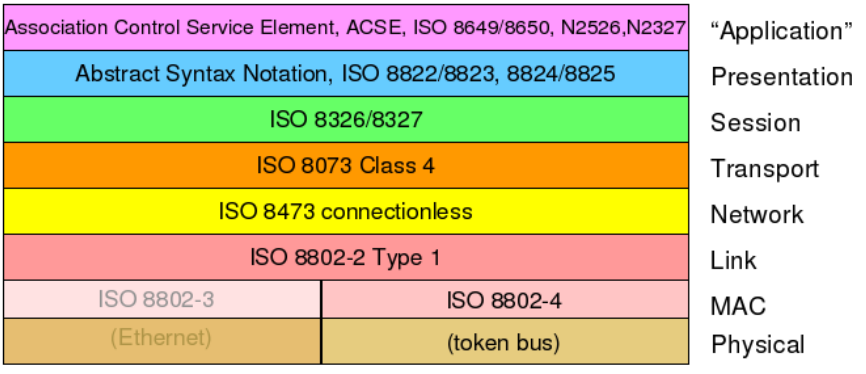

Fig. 3. The MMS communication stack specified as OSI layers

the exact octet sequences which are used to encode any given data item before it is transmitted over a network. The BER syntax is defined by the ITU-T's X.690 standards document, which is part of the ASN.1 document series.

BER is a self-identifying and self-delimiting encoding scheme, which means that each data value can be identified, extracted and decoded individually [6]. Each data element is encoded using a triplet consisting of a type identifier (tag), a length description and the actual data element. The use of such a triplet for encoding is commonly referred to as a tag-length-value (TLV) encoding. A generic triplet is depicted below.

```
[identifier (tag)] [length (of the contents)] [contents]
```

The use of TLV encoding allows any receiver to decode the ASN.1 information from an incomplete information stream, without any pre-knowledge of the size, content or semantic meaning of the data. Assuming that the communicating parties share the same *context specific* module definitions.

BER uses a unique code as an identifier for an ASN.1 data type. This identifier is encoded as one or more bytes of every data type and creates the *tag*. The identifier is well-structured to allow the representation of three levels of

Table 2. Description of the BER identifier

Bit number	7	6	5	4	3	2	1	0	Implication
	0	0							UNIVERSAL
	0	1							APPLICATION SPECIFIC
	1	0							CONTEXT SPECIFIC
	1	1							PRIVATE
			0						primitive data type
			1						constructed data type
				X	X	X	X	X	numeric identifier

information within one such code, as illustrated in table 2. On the highest level, represented by the highest-order two bits of the *tag octet(s)*, the class of the data type is encoded [6]. The third highest bit of the identifier indicates whether the represented data type is a primitive or constructed one. A constructed data type can be seen as a complex or compound data type hierarchically based on one or more primitive data types. The remainder of the identifier is a numeric tag associated with a data type within a class. Tags ranging from 0 to 30 can be associated with the remaining 5 bits of the octet. For larger tags, these 5 bits are set to 111111, and one or more subsequent octets are used to encode the tag.

4 Analysis of MMS Communication

ISO 8823 states that the OSI transport protocol exchanges information between peers in discrete units of information called Transport Protocol Data Units (TPDUs) [7]. This is a fundamental difference between the TCP and the network service expected by Transport Protocol Class 0 (TP 0). The difference is that TCP manages a continuous stream of octets, with no explicit boundaries, while TP0 expects information to be sent and delivered in discrete objects termed network service data units. Therefore RFC 1006 [8] describes that all TPDUs shall be encapsulated in discrete units called TPKTs. The TPKT layer, depicted in figure 4, is used to provide these discrete packets to the OSI Connection-Oriented Transport Protocol (COTP) on top of TCP.

We have intercepted some packages using Wireshark. As Wireshark did not support the MMS protocol at the time of testing, we were forced to manually decode the MMS PDUs. When looking at MMS communication in Wireshark we found the underlying MMS protocol stack depicted in 4.

As there was no publicly available documentation on how the vendor had implemented the MMS protocol we had to analyse the protocol stack. As we see in Fig. 4, there are two protocols running on top of TCP. Above TCP we find TPKT, which is a packet format used to emulate the ISO transport services COTP on top of TCP. RFC 1006 [8] describes how to implement ISO's transport protocol class 0 on top of TCP. ISO 9506 [1] stipulates the use of OSI transport class 4 in conjunction with MMS. Nevertheless RFC 1006 [8] describes the use of OSI transport class 0 to emulate an ISO Transport Service on top of the TCP.

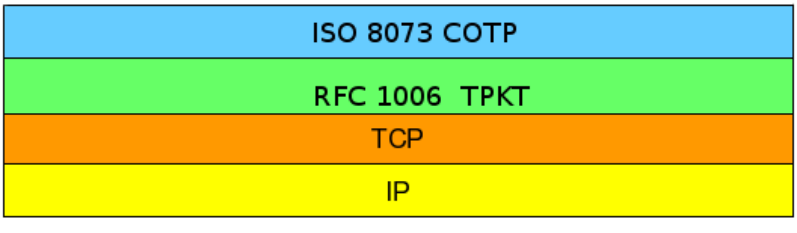

Fig. 4. The MMS communication stack as Wireshark detects it

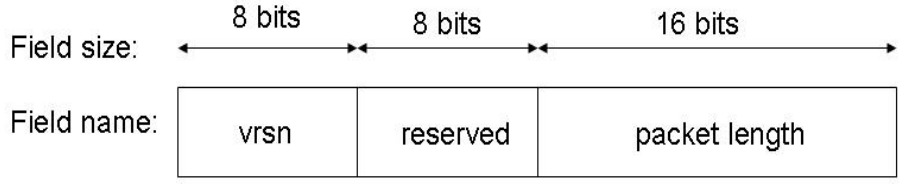

Fig. 5. Format of TPKT header

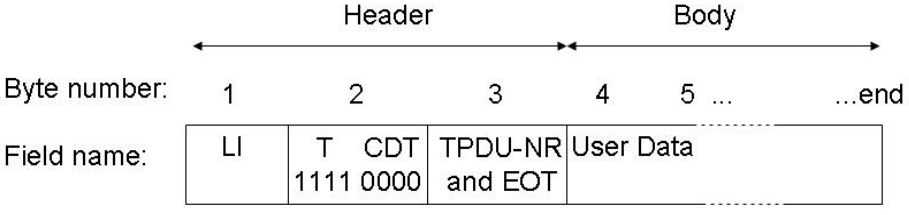

Fig. 6. Format of COTP PDU

The reason for using ISO's OSI transport class 0 on top of TCP/IP instead of transport class 4 is that transport class 0 achieves identical functionality as transport class 4 when running on top of TCP. The TCP layer provides reliable transport service through error detection and retransmission. It also handles segmentation and reassembly of PDUs. As TCP provides all these properties as part of its service to the next layer, there is no reason to implement them again.

A TPKT consists of two parts: a packet-header and a TPDU. The format of the header is constant regardless of the type of packet, as illustrated in Fig. 5.

The field labeled *vrsn* is the version number which according to RFC 1006 [8] always is three. The next field, *reserved*, is reserved for further use. The last field is the *packet length*. This field contains the length of entire packet in octets, including packet-header. The maximum TPDU size is 65531 octets, with a payload of maximum 65524 octets.

According to Wireshark we find COTP above the TPKT layer. RFC 0905 [9] describes the ISO 8073 specification. The COTP PDU is described in Fig. 6.

The header length in octets is indicated by a binary number in the *length indicator* (LI) field. This field has a maximum value of 254 (1111 1110)[3]. The next field is divided into two parts, first the PDU type specification (T), which describes the structure of the rest of the PDU, e.g., Data Transfer (1111) as described in Figure 6. The PDU type is encoded as a four bit word. The full list of codes for data types can be found in [9]. The second part, is the credit part (CDT) which is used to indicate a reliable transport service, but this is always set to 0000 as TP 0 does not offer reliable transport. The third field contains the TPDU number and an end of transfer indication flag. In all data transfer packets

[3] The value 255 (1111 1111) is reserved for possible extensions.

the EOT flag is set and the TPDU number is zero; this might be because the service relies on TCP sequence numbering on the transport layer, but we have not found any written documentation to support this theory.

We wish to note that there is no reference to ACSE in our packet dump. We verified through Wireshark's documentation that ACSE is a supported protocol [10]. That leaves us with two possible conclusions:

1. The MMS protocol uses an implementation of ACSE which is not in conformance with the standard, which leaves Wireshark unable to decode the packet layer.
2. The implementors of the MMS protocol have omitted the ACSE layer when implementing the protocol.

We are fully capable of decoding the whole payload of the COTP PDU to MMS structured ASN.1 text. We therefore conclude that the current implementation of the MMS has omitted the ACSE layer, making the ACSE authentication facilities forfeit. This means that there are no authentication or access control facilities at the lower layers of the MMS stack.

5 Decoding MMS Communication

Now knowing the underlying protocols which MMS is running on, we will study the MMS message communication between the MMS client and the MMS server and try to determine if there are any signs of security mechanisms. We used the client software to create a small program which we downloaded to the controller over MMS. The program was a very simple *counting application* as decribed in C code below:

```
int i=0;

while(TRUE)
    i=i+1;
```

The controller will now report the value of `i` back to the client at regular intervals using MMS. Once the program was downloaded to the controller and running, we used Wireshark to capture MMS communication on the network. The first thing we noticed when we examined the packet dump in Wireshark, was that there is a pattern in packet communication repeating itself in a period of eight. This pattern was first identified by packet sizes repeating themselves at a period of eight.

We wish to note that we chose an arbitrary packet in our packet dump as our starting point and decoded packages sequentially from that point. We chose this strategy to simulate an attacker tapping into a network at an arbitrary point in time.

5.1 Decoding the First PDU

We will now look closer at the first PDU and attempt to decode it. We have extracted the payload of the COTP package at our randomly chosen starting point. We know from the manufacturer's web page that the equipment employs MMS.

```
a0 41 02 01 7b a4 3c a1 3a a0 38 30 0c a0 0a
80 08 24 4d 53 47 24 31 24 24 30 15 a0 13 80
11 24 48 57 53 34 35 38 35 34 33 32 30 3a 4e
4f 52 4d 30 11 a0 0f 80 0d 24 4d 53 47 24 35
35 32 36 35 38 39 36
```

The package above is encoded in BERs TLV format. We know this from [2] and [3] which are whitepapers publicly available on the internet. We must therefor use the decoding rules described in section 3 to decode each TLV pair. When decoding this first PDU, with the help of the MMS syntax module[11], we found that this is a *confirmed-Request PDU*. This *confirmed-Request PDU* contains an integer id named *invokeID* with value 123 and *confirmedServiceRequest* for an read operation. The *read-request* specifies a *listOfVariables* with three items. Each item is a *vmd-specific object name* containing the *identifier*. We decoded these *identifiers* to:

- MSG1$$
- $HWS45854320:NORM
- MSG55265896

Space does not permit going through the entire decoding process, but for illustrtive purposes, the beginning of the first PDU is decoded below. Using table 2 we decode the first PDU to the following textual ASN.1 structure:

```
1
  a0                  CONTENT SPECIFIC constructed nr 0
3 41                  LENGTH=65

5   02                UNIVERSAL primitive nr 2 (INTEGER)
    01                LENGTH=1
7     7b              123

9   a4                CONTENT SPECIFIC constructed nr 4
    3c                LENGTH=60
11
      a1              CONTENT SPECIFIC constructed nr 1
13    3a              LENGTH=58

15      a0            CONTENT SPECIFIC constructed nr 0
        38            LENGTH=56
17
            30        UNIVERSAL constructed nr 16 (SEQUENCE)
```

```
19              0c          LENGTH=12

21              a0          CONTENT SPECIFIC constructed nr 0
                 0a         LENGTH=10
23
                 80         CONTENT SPECIFIC primitive nr 0
25               08         LENGTH=8

27                24        $
                  4d        M
29                53        S
                  47        G
31                24        $
                  31        1
33                24        $
                  24        $
35 ... etc.
```

We observe that MMS utilizes many *CONTENT SPECIFIC* tags to identify MMS specific data types. As stated earlier we can use the MMS module definition to decode these tags. This module is publicly available for anyone at [11] or through Google™.

6 Security in MMS

After analysing MMS protocol communications we will in this section look into the security mechanisms defined by the MMS standard. The standard does specify means for access control through *accessControlList* objects. We quote from the ISO standard [1]:

> The &accessMethod field for an Named Variable object shall specify the mode of access. If the Address is declarable (and obtainable) using MMS services, the &accessMethod field shall have the value public, and the Address attribute shall be defined and available to MMS clients requesting the attributes of the Named Variable object. Otherwise, the value of this field is a local issue. The public access method shall not be available unless vadr is supported.

From the quote above we see that each Named Object Variable has an *accessMethod* field, which specify the mode of access. According to the standard, if the address is declarable and obtainable using MMS service primitives the *accessMethod* shall have the value *public*. Access to all objects can be controlled by a special object, the Access Control List, that tells which client can read, delete or modify the object. On a general level MMS specifies that if the accessMethod is public the following field shall appear and if the &accessMethod is anything but public, the following field shall not appear. But there are some exceptions.

An MMS server may declare an MMS variable that exists only at the instant of access. Such a variable does not have an address per se, but is still accessible at that instant. We see that the standard provides mechanisms for access control, but to our knowledge there are no other security mechanisms included in the MMS standard. We quote from the Security Considerations section in the ISO standard [1]:

> When implementing MMS in secure or safety critical applications, features of the OSI security architecture may need to be implemented. This International Standard provides simple facilities for authentication (passwords) and access control. Systems requiring a higher degree of security will have to consider features beyond the scope of this International Standard. This International Standard does not provide facilities for non-repudiation.

As stated above, MMS itself is not designed with information security in mind. This indicates that security should be enforced at some lower layer, but as we have seen through our analysis of the MMS, there is no security enforced at any layer. The ACSE layer could have offered some security features, but as the ACSE layer is omitted from our implementation those features are forfeit.

According to the ISO standard the MMS protocol should have implemented some simple facilities for password authentication and access control. We wanted to study these mechanisms to see what security they really offer, but as they are at best optional and at worst not implemented they provide no security what so ever. We have through our analysis seen that they do not exsist.

When analysing the MMS we have found no protection against replay attacks. This is a major concern, as anyone with access to the network may sniff up a packet and then replay it on the network at a an inappropriate moment. We do not regard the *invokeID* field as such a mechanism as it is easily changed.

7 Conclusion

The Manufacturing Messaging Protocol (MMS) is a complex protocol that is rendered even more complex by the implementation of an OSI transportation protocol on top of TCP. MMS offers very limited security mechanisms, and equipment we have studied does not appear to have implemented even these mechanisms. It is clear that if MMS is to be used in process control networks that have to fulfil information security requirements, major modifications have to be made to the protocol.

Acknowledgements

The research for this paper was conducted while Mr. Sørensen was a student at the Norwegian University of Science and Technology (NTNU).

References

1. Industrial automation systems, Manufacturing Message Specification. Part 1, ISO ISO Standard ISO 9506-1:2003(E) (2003)
2. Overview and introduction to the Manufacturing Message Specification (MMS), System Integration Specialists Company (SISCO), 6605 19,5 Mile Road, Sterling Heights, MI 48314-1408, USA, Tech. Rep. (1995), http://www.sisconet.com/downloads/mmsovrlg.pdf
3. Falk, H., Robbins, J.: An Explanation of the Architecture of the MMS standard. System Integration Specialists Company (SISCO), 6605 19, 5 Mile Road, Sterling Heights, MI 48314-1408, USA, Tech. Rep. (1995)
4. Falk, H., Burns, D.M.: MMS and ASN.1 Encoding. System Integration Specialists Company (SISCO), 6605 19, 5 Mile Road, Sterling Heights, MI 48314-1408, USA, Tech. Rep. (2001)
5. Floyd, L., Ronald, D.: Manufacturing automation protocol. In: Conference Record - International Conference on Communications, pp. 620–624 (1985)
6. Basic encoding rules, Vijay Mukhi's Computer Institute, India (February 2007), http://www.vijaymukhi.com/vmis/ber.htm
7. Information technology – Open Systems Interconnection – Connection-oriented presentation protocol: Protocol specification. ISO ISO Standard ISO/IEC 8823-1:1994 (1994)
8. Rose, M.T., Cass, D.E.: RFC 1006: ISO transport services on top of the TCP: Version 3 (May 1987), obsoletes RFC0983. Updated by RFC 2126. Status: STANDARD, ftp://ftp.internic.net/rfc/rfc1006.txt
9. McKenzie, A.M.: RFC 905: ISO transport protocol specification ISO DP 8073 (April 1984), ftp://ftp.internic.net/rfc/rfc905.txt
10. Wireshark wiki on ACSE, Published on Wireshark's wiki page (June 2006), http://wiki.wireshark.org/ACSE
11. SISCO's MMS syntax, Published on Systems Integration Specialists Company (SISCO), Inc. Homepage (February 2007), http://www.sisconet.com/downloads/mms_abstract_syntax.txt